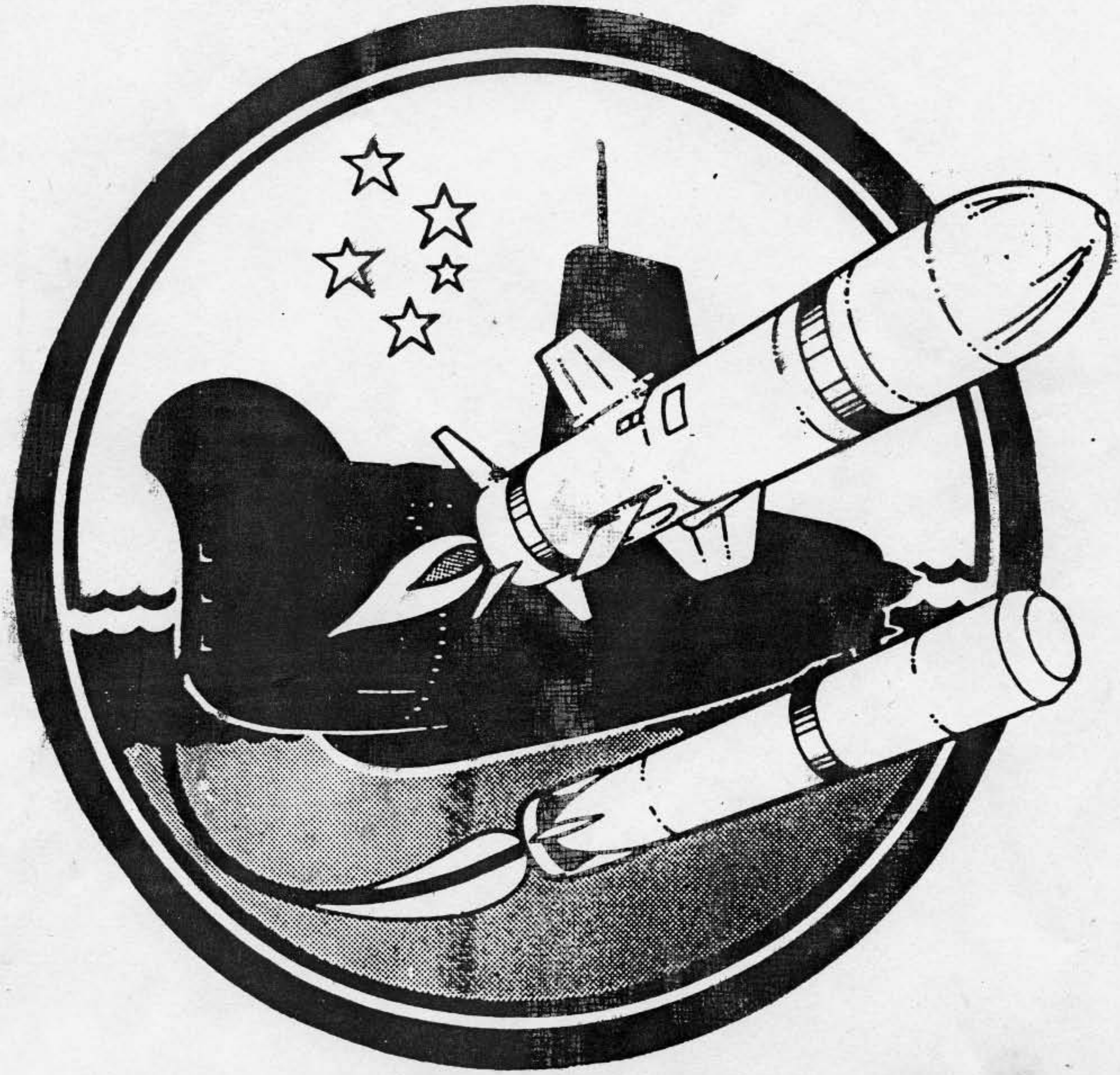
ROYAL AUSTRALIAN NAVY
SUBMARINE SCHOOL
KNOWLEDGE IN DEPTH



DIGITAL THEORY REFRESHER

INTRODUCTION

GENERAL

Digital logic is related to rational thought processes of the mind, where we express our decisions by talking, writing, or action. Digital systems outputs are electrical signals that can perform a multitude of functions.

It is often stated that digital logic has developed from philosophical and mathematical logics. It is this interrelationship that is suggesting more formal, versatile, and powerful ways in which to analyse and utilise digital circuits.

DIGITAL AND ANALOG COMPARISONS

The prime difference between the two functions is that analog refers to how much, whilst digital is interested in how many. Analog is a continually changing process with infinite variables. Digital, however, is a process of discrete definitive values.

An example of an analog is the range of temperatures between 26°C and 27°C, the only restriction is that of precision. Cash is an illustration of discrete digital steps. When it is counted the result is a precise amount, with the smallest possible step limited by the denomination of the least valuable coin.

ORIGIN OF DIGITAL SYSTEMS

Digital electronics began when the first person learned to count, learned to associate number names with objects in a group. Most counting was done on the fingers (digits), and for this reason the basic number names (one, two, three .....) are known as DIGITS.

The invention of numbers led to arithmetic and all kinds of calculating devices like the abacus, Napier's bones (the first slide rule), and Pascal's calculator (the first adding machine). But the really crucial inventions in the evolution of digital electronics were made in the nineteenth century.

To begin with, Jacquard (1801) invented an automatic loom whose main feature was the use of PUNCHED CARDS. In Jacquard's loom needles passed through the holes in such a card and stitched a pattern onto cloth. By using cards with different hole patterns, Jacquard could produce all kinds of figures easily and reliably.

In 1833, Babbage visualized the first COMPUTER, a machine that used punched cards to carry out arithmetic calculations automatically. By a prearranged code, certain groups of holes in these cards were to represent either numbers or instructions. The key idea in Babbage's computer was to enter all numbers and instructions before the calculation began; then on command, the computer was to carry out all the steps in the calculation without human intervention. (This is the crucial difference between a calculator and a computer. A calculator depends on human intervention because someone has to enter numbers and instructions while the calculation is in progress.)

In 1854 Boole found a new way of thinking, a new way to reason things out. He decided to use symbols instead of words to reach logical conclusions. Boole saw a pattern in the way we think that allowed him to invent symbolic logic, a method of reasoning based on the manipulation of letters and symbols. In many ways, symbolic logic resembles ordinary algebra. This system has been called BOOLEAN ALGEBRA.

Although originally intended for solving logic problems, Boolean algebra now finds its greatest use in the design of digital computers. By a coincidence, the rules of symbolic logic apply to the electronic circuits in computers and other digital systems.

Babbage never built a working model of a digital computer, but his notes prove he knew how to go about it. His ideas opened up a whole new world and led to today's modern computers.

The first electronic computers based on Babbage's ideas appeared in the early 1950s. These FIRST GENERATION computers used vacuum tubes. Toward the end of the same decade, SECOND GENERATION computers were developed. (They used transistors.) In the early 1960s THIRD GENERATION computers evolved; these used transistors and some integrated circuits. We're now in the FOURTH GENERATION of computers; these make extensive use of integrated circuits and microprocessor devices.

1.4 USES OF DIGITAL SYSTEMS

Common use of digital systems, apart from industrial process control and international satellite communications, includes typewriters that display-replay-modify and copy, weight measurement-unit price-total cost scales, electronic measuring devices, vending and poker machines, banking/betting/travel and reservation networks, cash registers with full inventory control abilities, and high fidelity multitrack recordings.

Military digital systems are used for cryptography, weapons selection/aim/fire control, navigation, high speed secure data links and machinery control.

NUMBERING SYSTEMS

GENERAL

In science, technology, business, and, in fact, in most other fields of endeavour, we are constantly dealing with QUANTITIES. These quantities are measured, monitored, recorded, manipulated arithmetically, observed, or in some other way utilized in most physical systems. It is important when dealing with various quantities that we be able to represent their values efficiently and accurately.

NUMERICAL REPRESENTATIONS

There are basically two ways of representing the numerical value of quantities: ***ANALOG*** and *DIGITAL*. We will only consider the digital method.

A digital system is a combination of devices (electrical, mechanical, photoelectric, etc.) arranged to perform certain functions in which quantities are represented digitally. Some of the more common digital systems are digital computers and calculators, digital voltmeters, and numerically controlled machinery. In these systems the electrical and mechanical quantities change only in discrete steps.

Generally speaking, digital systems offer the advantages of greater speed and accuracy and the capability of memory. In addition, digital systems are generally more versatile in a wider range of applications.

Many number systems are in use in digital technology. The most common are the ***DECIMAL***, *BINARY*, *OCTAL* and ***HEXIDECIMAL*** systems.

DECIMAL SYSTEM

The decimal system is composed of 10 numerals or symbols, which are commonly referred to as *DIGITS*. These 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9; using these symbols we can express any quantity. The decimal system is also called the ***BASE – 10*** system because it has 10 digits. The decimal system is a ***POSITIONAL – VALUE*** system in which the value of a digit depends on its position. For example, consider the decimal number 358. We know that the digit 3 actually represents 3 HUNDREDS, the 5 represents 5 TENS, and the 8 represents 8 UNITS. In essence, the 3 carries the most weight of the three digits; it is referred to as the ***MOST SIGNIFICANT DIGIT (MSD)***. The 8 carries the least weight and is called the *LEAST SIGNIFICANT DIGIT (LSD)*.

Consider the following example, 7569_{10} :–

BASE & POWER →	10^3	10^2	10^1	10^0
VALUE →	1000	100	10	1
NUMBER →	7	5	6	9
	7×10^3	5×10^2	6×10^1	9×10^0
	7000+	500+	60+	9

$= 7569_{10}$

BINARY SYSTEMS

Unfortunately, the decimal system does not lend itself to convenient implementation in digital systems, however, it is very easy to design simple, accurate electronic circuits that operate with only two voltage levels. For this reason, almost all digital systems use the Binary number system *(base 2)* as the basic number system of its operations, although other systems are often used in conjunction with binary.

In the binary system there are only two symbols or possible digit values, 0 and 1. Even so, this base – 2 system can be used to represent any quantity that can be represented in decimal or other number system.

Consider the following example, 11011_2 :–

2^4	2^3	2^2	2^1	2^0
16	8	4	2	1
1	1	0	1	1
1 x 16	1 x 8	0	1 x 2	1 x 1
16+	8+	0+	2+	1

$= 27_{10}$

OCTAL SYSTEM

Is a base 8 number system, and uses only the digits 0 to 7. Consider the following example, 7364_8 :–

8^3	8^2	8^1	8^0
512	64	8	1
7	3	6	4
7×8^3	3×8^2	6×8^1	4×8^0
3584+	192+	48+	4

$= 3828_{10}$

NOTE: The base of the number is usually included to distinguish between the number systems in use.

HEXIDECIMAL SYSTEM

Is a base 16 system, and uses the numbers 0 to 15, where 10 to 15 are substituted by the letters A to F. ie. the numbers are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E & F.

Consider the following example, 1C9F:–

16^3	16^2	16^1	16^0
4096	256	16	1
1	C	9	F
1×16^3	12×16^2	9×16^1	15×16^0
4096 +	3072 +	144 +	15

$= 7327_{10}$

FRACTIONS

eg. 0.5954_{10}

10^{-1}	10^{-2}	10^{-3}	10^{-4}
0.1	0.01	0.001	0.0001
5	9	5	4
5×10^{-1}	9×10^{-2}	5×10^{-3}	4×10^{-4}
0.5 +	0.09 +	0.005 +	0.0004

$= 0.5954_{10}$

eg. 0.0110_2

2^{-1}	2^{-2}	2^{-3}	2^{-4}
0.5	0.25	0.125	0.0625
0	1	1	0
0 +	0.25 +	0.125 +	0

$= 0.375_{10}$

NOTE: The above is true for all base systems.

CONVERSION OF BASES

DECIMAL TO BINARY

eg. 231.845_{10} to base 2.

÷ 2

divide the integer by the new base. Any remainder to right of line (Do not divide into remainder.)

231		.	845
115	1	1	690
57	1	1	380
28	1	0	760
14	0	1	520
7	0	1	040
3	1		
1	1		
0	1		

x 2

multiply fraction by new base, spill over to the left of line (Do not x spillover.)

Binary number read clock-wise from bottom left.

$231.845_{10} = 11100111.11011_2$

DECIMAL TO OCTAL

eg. 28.85_{10} to base 8.

÷ 8

28		.	85
3	4	6	80
0	3	6	40
		3	20

x 8

ie. $28.85_{10} = 34.663_8$

DECIMAL TO HEXIDECIMAL

eg. 3639_{10} to base 16.

÷ 16

3639			
227	7	=	7
14	3	=	3
0	14	=	E

ie. $3639_{10} = E37_{16}$

BINARY TO OCTAL

(by groups of three (3) FROM the decimal point.)

eg. 1 1 0 1 0 0 1 . 1 1 0 1 1 1 $_{(2)}$

0 0 1	1 0 1	0 0 1	.	1 1 0	1 1 1	(2)
1	5	1	.	6	7	(8)

Then 1101001 . $110111_{(2)}$ = 151 . 67 $_{(8)}$

BINARY TO HEXIDECIMAL

(by groups of four (4) FROM the decimal point.)

eg. 1 1 1 0 0 0 0 0 1 0 . 1 0 0 0 1 1 $_{(2)}$

0 0 1 1	1 0 0 0	0 0 1 0	.	1 0 0 0	1 1 0 0	(2)
3	8	2	.	8	C	(16)

Then 1110000010100011 $_{(2)}$ = 382 . 8C $_{(16)}$

NOTE: For conversions from OCTAL or HEXIDECIMAL to Binary the inverse is also true.

i.e. for each octal number, replace with groups of three (3) binary BITS to the decimal equivalent of that number.

For each hexidecimal, replace with a group of four (4) binary BITS to the decimal equivalent of that number.

eg.

OCTAL 5 3 7 . 4

= 101	011	111	.	100	(2)
5	3	7	.	4	

HEX. A D . F

= 1010	1101	.	1111	(2)
A	D		F	

Appendix A lists a table of base conversions of 0_{10} to 255_{10} to Binary, Octal, and Hexidecimal. Appendix C lists a table of powers of 2 (positive and negative powers are both listed).

BOOLEAN ALGEBRA

BOOLEAN ALGEBRA

Boolean Algebra is a very simple form of algebra that describes logical switching functions. It is well suited to analysis, fault finding and design of digital circuitry because of its ability to express all logic functions as '1' or '0'.

The reason for Boolean Algebra is primarily to simplify a complicated logic circuit to a simple logic circuit.

Boolean Expressions can be derived from logic diagrams, ie.

a. Begin with the left of the diagram and find the output expression for each logic element.

b. An input expression to any element may be represented by two or more letters. These letters should remain grouped in the output expression.

Fig. 3.1 shows how this is carried out.

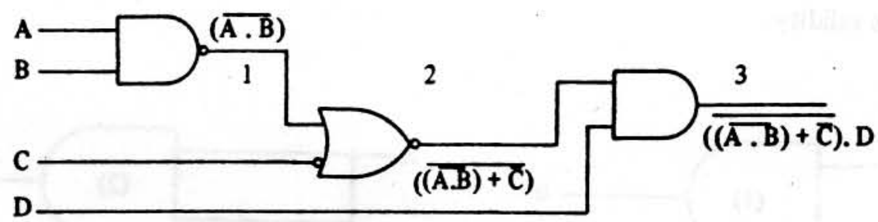


Fig. 3.1 Deriving a Boolean Expression from a Logic Circuit

Logic diagrams can also be derived from Boolean Expressions, ie.

a. Begin by constructing the diagram at the right and work to the left until all of the inputs become single letters.

b. Never separate the letters in a group until the group has been separated from the other groups in the expression.

c. If the Viniculum (BAR) extends over more than one letter use an inverter to remove it.

Consider the example shown in Fig. 3.2.

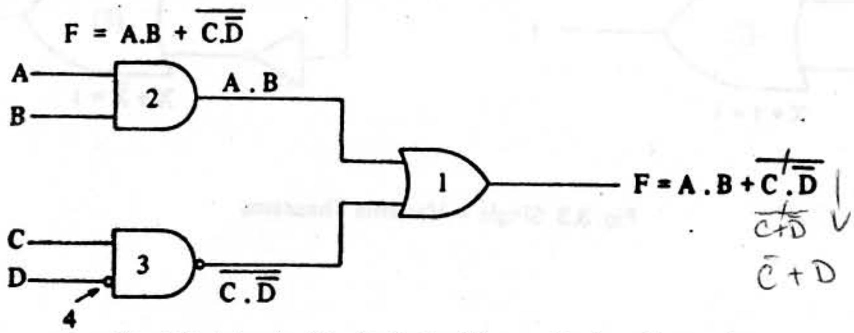


Fig. 3.2 A Logic Circuit Derived from a Boolean Expression

TRUTH TABLE

A Truth Table is a table that shows all the input and output possibilities for a logic circuit. In other words it uniquely defines the operation of a logic circuit.

A Truth Table must be used for the maximum number of combinations possible, using 2^n, where n = the number of input variables, ie.,

2 i/p = 2^2 = 4 possible combinations

3 i/p = 2^3 = 8 " "

4 i/p = 2^4 = 16 " "

Boolean Expressions can be extracted from the Truth Table by:

a. Noting which combination of inputs give a '1' output.

b. Recognising that each '1' output is the result of an AND function.

c. OR-ing all the AND functions to arrive at the Boolean Expression.

d. Reduce the Boolean Expression to its simplest terms.

BOOLEAN THEOREMS

The first group of theorems is given in Fig. 3.3. In each theorem X represents a logic variable that can be either 0 or 1. Each theorem is accompanied by its equivalent logic circuit to help verify its validity.

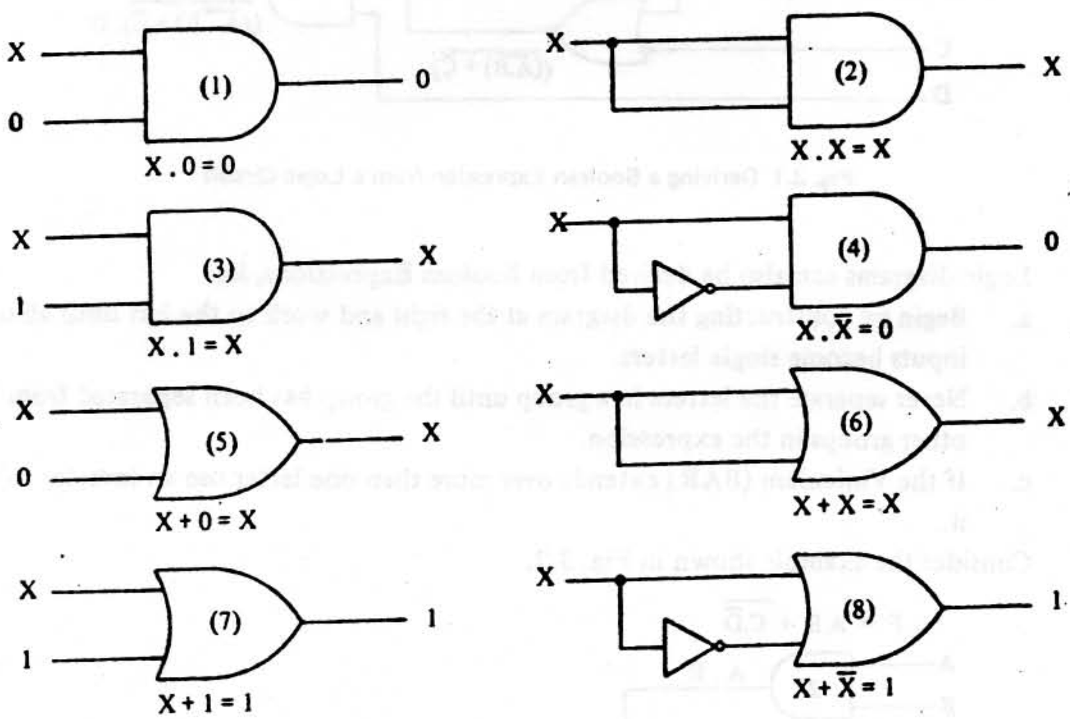


Fig. 3.3 Single – Variable Theorems

BASIC LAWS AND COMMON IDENTITIES OF BOOLEAN ALGEBRA

MULTIPLICATION	:	$(A + B)\ (A + B)$	or	$AA + AB + BA + BB$
COMMUTATIVE	:	$AB = BA$	or	$A + B = B + A$
ASSOCIATIVE	:	$A(BC) = ABC$	or	$A + (B + C) = A + B + C$
INDEMPOTENT	:	$AA = A$	or	$A + A = A$
DOUBLE NEGATIVE	:	$\overline{\overline{A}} = A$		
COMPLEMENTARY	:	$\overline{A}A = 0$	or	$\overline{A} + A = 1$
INTERSECTION	:	$A \,.\, 1 = A$	or	$A \,.\, 0 = 0$
UNION	:	$A + 1 = 1$	or	$A + 0 = A$
DE MORGAN'S THEOREM	:	$\overline{AB} = \overline{A} + \overline{B}$	or	$\overline{A + B} = \overline{A} \,.\, \overline{B}$
DISTRIBUTIVE	:	$A(B + C) = AB + AC$	or	$A + (BC) = (A+B)(A+C)$
ABSORBTION	:	$A(A + B) = A$	or	$A + (AB) = A$
COMMON IDENTITIES	:	$A(\overline{A} + B) = AB$	or	$A + \overline{A}B = A + B$
UNNAMED LAWS	:	$\overline{A} + AB = \overline{A} + B$	or	$\overline{A} + A\overline{B} = \overline{A} + \overline{B}$

The following is an example of how a Boolean Expression may be taken from a Truth Table, simplified, (using the laws given above) and the simplified circuit drawn from the new expression.

A	B	C	F	
0	0	0	0	
0	0	1	1	$= \overline{A}.\overline{B}.C +$
0	1	0	0	
0	1	1	0	
1	0	0	1	$= A.\overline{B}.\overline{C} +$
1	0	1	1	$= A.\overline{B}.C +$
1	1	0	0	
1	1	1	1	$= A.B.C$

Output expression $= \overline{A}.\overline{B}.C + A.\overline{B}.\overline{C} + A.\overline{B}.C + A.B.C$

Simplified to $A.C + \overline{B}.C + A.\overline{B}$.

Simplification process shown on the following page.

$\bar{A}.\bar{B}.C + A.\bar{B}.\bar{C} + A.\bar{B}.C + A.B.C =$

$\bar{B}(\bar{A}.C + A.\bar{C} + A.C) + A.B.C$

$\bar{B}(\bar{A}.C + A(\bar{C} + C)) + A.B.C$ $\quad \bar{C} + C = 1$ Complimentary

$\bar{B}(\bar{A}.C + A.1) + A.B.C$ $\quad A . 1 = A$ Intersection

$\bar{B}(\bar{A}.C + A) + A.B.C$ $\quad \bar{A}.C + A = A{+}C$ Common Identities

$\bar{B}(A + C) + A.B.C$

$\bar{B}.A + \bar{B}.C + A.B.C$

$\bar{B}.A + C(\bar{B} + A.B)$ $\quad \bar{B} + A.B = \bar{B} + A$ Unnamed Law

$\bar{B}.A. + C(\bar{B} + A)$ ANS

or

$A.\bar{B} + \bar{B}.C + A.C$ ANS

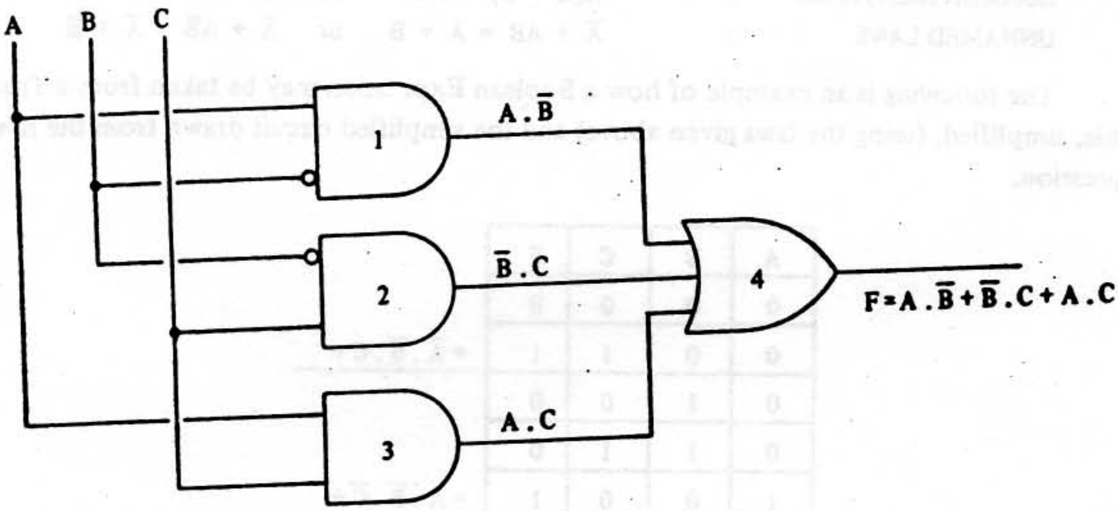


Fig. 3.4 Simplified Circuit for the Expression found in the Previous Example

CONVERTING FROM EXPLICIT LOGIC TO IMPLICIT LOGIC

a. To convert OR gates to NOR gates, replace all OR's with NOR's and invert the outputs.

b. To convert AND gates to NOR gates, replace all AND's with NOR's and invert all inputs.

c. To convert AND gates to NAND gates, replace all AND's with NAND's and invert the outputs.

d. To convert OR gates to NAND gates, replace all OR's with NAND's and invert all inputs.

Exercise, convert the circuit in Fig. 3.4 to IMPLICIT logic using all NAND logic.

VIETCH DIAGRAMS

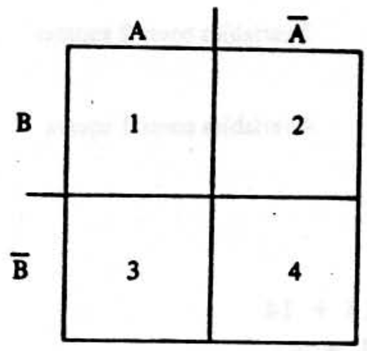
A simpler method of reducing Boolean Expressions can be performed by using a Vietch Diagram.

For two variables there are four miniterms (variables). $2^2 = 4$ sq.

For three variables there are eight miniterms. $2^3 = 8$ sq.

For four variables there are 16 miniterms. $2^4 = 16$ sq.

Two Variables – Four Squares

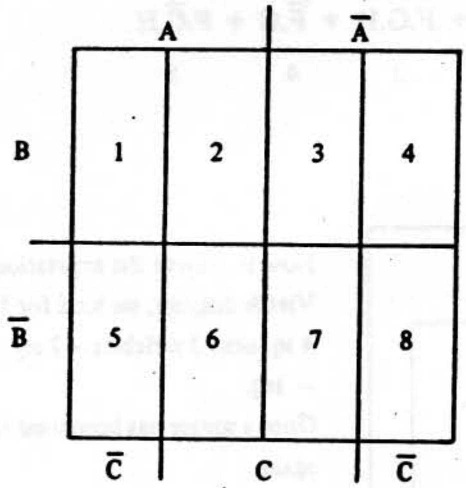


1 variable covers 2 squares

2 variables cover 1 square

eg. $A = 1 + 3$; $B = 1 + 2$; $\overline{A} = 2 + 4$; $\overline{B} = 3 + 4$

Three Variables – Eight Squares



1 variable covers 4 squares

2 variables cover 2 squares

3 variables cover 1 square

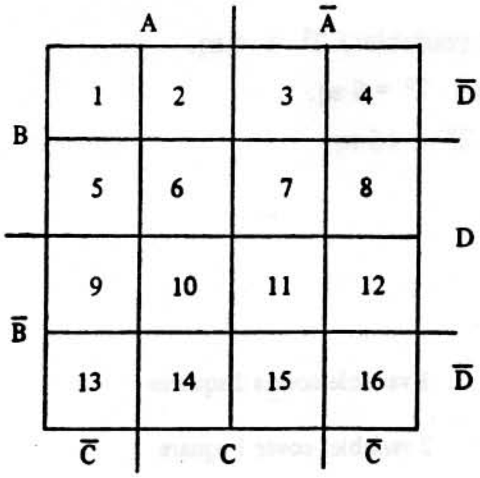
eg. $A = 1 + 2 + 5 + 6$; $B = 1 + 2 + 3 + 4$; $C = 2 + 3 + 6 + 7$

$\overline{A} = 3 + 4 + 7 + 8$; $\overline{B} = 5 + 6 + 7 + 8$; $\overline{C} = 1 + 5 + 4 + 8$

$A.B = 1 + 2$; $\overline{A}.\overline{C} = 4 + 8$; $\overline{B}.C = 6 + 7$

$A.\overline{B}.\overline{C} = 5$; $\overline{A}.\overline{B}.\overline{C} = 8$; $A.B.C = 2$

Four Variables – 16 Squares



1 variable covers 8 squares

2 variables cover 4 squares

3 variables cover 2 squares

4 variables cover 1 square

eg. $A = 1 + 2 + 5 + 6 + 9 + 10 + 13 + 14$

$D = 5 + 6 + 7 + 8 + 9 + 10 + 11 + 12$

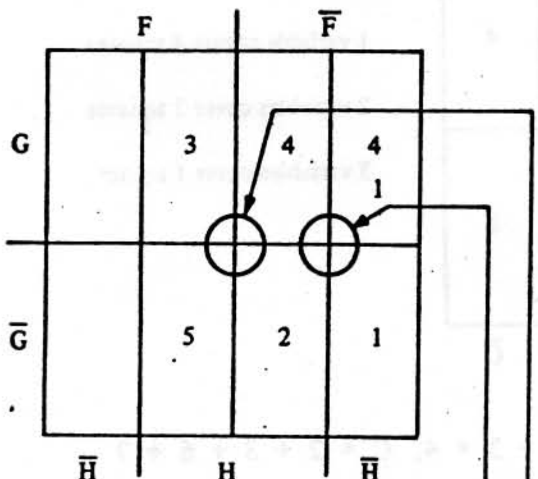
$A.B = 1 + 2 + 5 + 6;$ $\overline{A}.D = 7 + 8 + 11 + 12$

$\overline{A}.C.\overline{D} = 3 + 15;$ $A.B.\overline{C} = 1 + 5$

$A.B.C.D = 6;$ $\overline{A}.B.\overline{C}.D = 8$

Example, put the following Boolean Expression into a Vietch diagram.

$$\underset{1}{\overline{F}.\overline{H}} + \underset{2}{\overline{F}.\overline{G}.H} + \underset{3}{F.G.H} + \underset{4}{\overline{F}.G} + \underset{5}{F.\overline{G}.H}$$



Now to remove the expression from the Vietch diagram, we look for 1 variable – 4 sq. next 2 variables – 2 sq., 3 variables – 1sq.

Once a square has been used it may be used again.

1) = $\overline{F}$

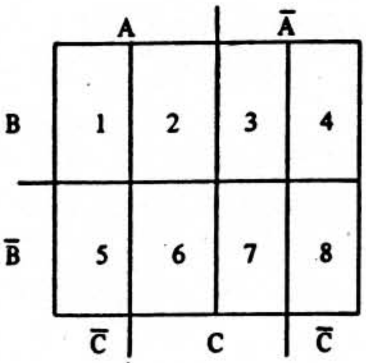
2) = H

ANS = $\overline{F}$ + H

ie. $\overline{F}.\overline{H} + \overline{F}.\overline{G}.H + F.G.H + \overline{F}.G + F.\overline{G}.H = \overline{F} + H$

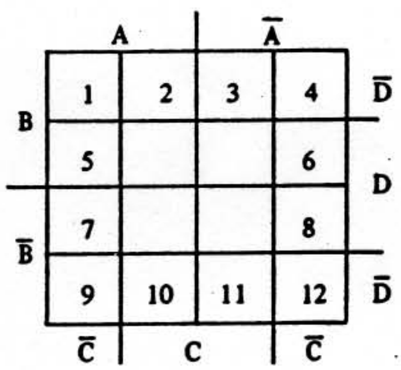
It should be noted that the Vietch diagrams can be thought of and treated like a cylinder, (ie. it can be rolled such that the left hand edge comes into contact with the right hand edge, or such that the top edge meets the bottom edge, thus forming more adjacent squares, should these squares be occupied.

As an example, consider the following three variable Vietch diagram.



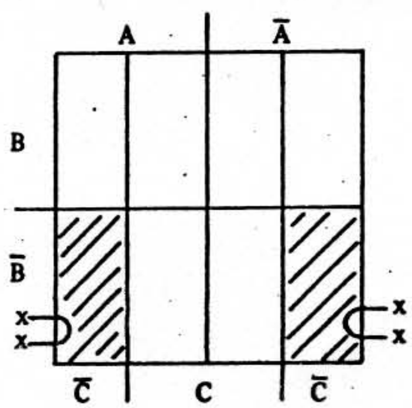
In this example, squares 1, 5, 4 & 8 are all adjacent squares and these represent $\bar{C}$. Similarly squares 4 & 1 = $B.\bar{C}$, and 5 & 8 = $\bar{B}.\bar{C}$

For a four variable Vietch diagram the same rules apply, eg.



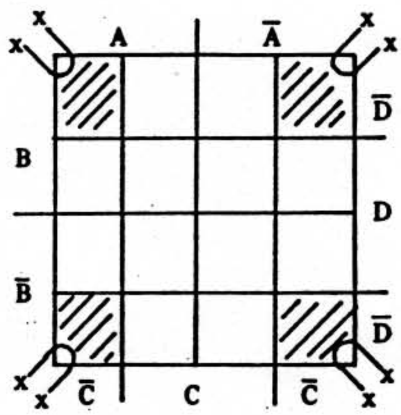
Squares on side 1, 5, 7 & 9 mate with their opposite square on side 4, 6, 8 & 12, whilst the squares 1, 2, 3 & 4 mate with squares 9, 10, 11 & 12.

This may be more clearly shown in the following examples.



(a)

= $\bar{B}.\bar{C}$



(b)

= $\bar{C}.\bar{D}$

BINARY ARITHMETIC OPERATIONS

BINARY ARITHMETIC

The addition of two binary numbers is performed in exactly the same manner as the addition of decimal numbers. In fact, binary addition is simpler since there are fewer cases to learn. Let us first review decimal addition:

$$\begin{array}{r} 3\;7\;\boxed{6} \leftarrow \boxed{\text{LSD}} \\ +\;4\;6\;1 \\ \hline 8\;3\;7 \end{array}$$

The least – significant – digit (LSD) position is operated on first producing a sum of 7. The digits in the second position are then added to produce a sum of 13, which produces a CARRY of 1 into the third position. This produces a sum of 8 in the third position.

The same general steps are followed in binary addition. However, there are only four cases that can occur in adding the two binary digits (bits) in any position. They are:

$$0 + 0 = 0$$
$$1 + 0 = 1$$
$$1 + 1 = 0 + \text{carry of 1 into next position}$$
$$1 + 1 + 1 = 1 + \text{carry of 1 into next position}$$

The last case occurs when the two bits in a certain position are 1 and there is a carry from the previous position. Some examples of binary addition are:

$$\begin{array}{r} 0\;1\;1\;(3) \\ +\;\;1\;1\;0\;(6) \\ \hline 1\;0\;0\;1\;(9) \end{array} \qquad \begin{array}{r} 1\;0\;0\;1\;(9) \\ +\;\;1\;1\;1\;1\;(15) \\ \hline 1\;1\;0\;0\;0\;(24) \end{array} \qquad \begin{array}{r} 1\;1\;.\;0\;1\;1\;(3.375) \\ +\;1\;0\;.\;1\;1\;0\;(2.750) \\ \hline 1\;1\;0\;.\;0\;0\;1\;(6.125) \end{array}$$

It is not necessary to consider the addition of more than two binary numbers at a time because in all digital systems the circuitry that actually performs the addition can only handle two numbers at a time. When more than two numbers are to be added, the first two are added together and then their sum is added to the third number; and so on.

Addition is the most important arithmetic operation in digital systems. As we shall see, the operations of subtraction, multiplication and division as they are performed in most modern digital computers and calculators actually use only addition as their basic operation.

Subtraction in the binary system is not as simple an operation as it is in the decimal system. The actual subtraction operation as shown below is not the method by which a digital circuit carries out the operation. The operation is, however, worth mentioning here.

eg. $14_{10} - 12_{10}$

$$\begin{array}{r} 14_{10} \\ -\;12_{10} \\ \hline +\;\;2_{10} \end{array} \qquad \begin{array}{r} 1\;1\;1\;0_2 \\ -\quad 1\;1\;0\;0_2 \\ \hline +\quad 0\;0\;1\;0_2 \end{array}$$

The rules for binary subtraction can be best summarised in the following table:

	A minus	B	DIFFERENCE	BORROW
	0	0	0	0
	0	1	1	1
	1	0	1	0
	1	1	0	0
extra	1 – (1+1)		1	1
rules	0 – (1+1)		0	1

Subtraction then, as shown above, is therefore quite a difficult operation to implement using digital circuits. A method then must be found which will allow us to carry out this operation using simpler digital circuits.

Two such methods which both use addition as the basis of the operation are: 1's COMPLEMENT, and the 2's complement.

• **1'S COMPLEMENT**

The 1's complement form of any binary number is obtained by changing every 0 in the number to a 1, and every 1 in the number to a 0. For example, the 1's complement of 1011001 is 0100110, and the 1's complement of 00111001 is 11000110.

Thus far we have considered only UN-signed (TRUE–MAGNITUDE) and since most digital calculators and computers handle negative as well as positive numbers, some means is required for representing the SIGN of the number (+ or –). This is usually done by adding another bit to the number called the SIGN bit.

When negative numbers are represented in 1's complement form, the sign bit is made a 1 and the magnitude is converted from true binary form to its 1's complement. To illustrate, the number –45 would be represented as follows:

sign bits

$$-45 = \boxed{1}\ 1\ 0\ 1\ 1\ 0\ 1 \qquad \text{(true magnitude form)}$$
$$= \boxed{1}\ 0\ 1\ 0\ 0\ 1\ 0 \qquad \text{(1's complement form)}$$

Note that the sign bit is not complemented but is kept as a 1 to indicate a negative number. Modern computers however use the most significant bit of a number to denote the number's sign as well as it being a part of the number. The number –45 then would appear as:

sign bits $\longrightarrow$

$$\boxed{1}\ 0\ 1\ 0\ 0\ 1\ 0 \qquad \text{(already 1's complement)}$$
$$-45 = \boxed{0}\ 1\ 0\ 1\ 1\ 0\ 1 \qquad \text{(true magnitude but sign bit from original form denotes a negative number)}$$

Subtraction of binary numbers using 1's complement would be carried out as follows:

45		0 1 0 1 1 0 1	(Minuend)
−14	−	0 0 0 1 1 1 0	(Subtrahend)
31		1 1 1 0 0 0 1	(1's Complement)
	+	0 1 0 1 1 0 1	(add Minuend)
Carry out or Spill		1 0 0 1 1 1 1 0	
(End Around Carry)		1	
		0 0 1 1 1 1 1	answer 31.

Note end around carry goes to the least significant bit position – including binary (decimal) place.

A variation occurs when the Subtrahend is larger than the Minuend (subtracting a larger number from a smaller number). In such cases there will be no overflow (carry out). The answer must now be complemented and called negative.

eg.	23		0 1 0 1 1 1	
	−25	−	0 1 1 0 0 1	
	− 2		1 0 0 1 1 0	1's complement
			0 1 0 1 1 1	Add the Minuend
Note: No Spill			1 1 1 1 0 1	To be complemented
			0 0 0 0 1 0	Correct answer −2.

2's COMPLEMENT

Since most modern computers use the 2's complement method for binary subtraction or addition of SIGNED numbers, it is important that we now discuss this method.

The 2's complement of a binary number is found by first obtaining the ones complement and then adding 1 to the least significant bit (L.S.B.) eg.

To find the 2's complement of		1 0 0 1 1 1
Obtain the 1's complement	=	0 1 1 0 0 0
Add 1 to the L.S.B.	−	1
The 2's complement then	=	0 1 1 0 0 1

Note: The Most Significant Bit (M.S.B.) still represents the sign of the number, where 1 = −ve, and 0 = +ve. The M.S.B. is still treated as part of the whole number and must therefore also be operated upon. Hence the number in the above example is negative (MSB = 1) and its true magnitude, found by 2's complementing is 39 decimal; ie. $100111_{(2)} = 39_{(10)}$.

Another (and quicker) method for finding the 2's complement of a binary number is: starting from the L.S.B., write down all the bits up to and including the first 1, thereafter complement (invert) the remaining bits, eg.

The 2's complement of	1 0 0 1	is	0 1 1 1
" " " "	0 1 1 0	"	1 0 1 0
" " " "	1 1 0 0	"	0 1 0 0
" " " "	1 0 0 0	"	1 0 0 0

ADDITION OF TWO'S COMPLEMENT NUMBERS

a. The addition of two positive numbers is straightforward. Consider the addition of +9 and +4 (both decimal numbers).

```
              sign bits ──┐
                          ↓
        +9           0 1 0 0 1    (augend)
        +4           0 0 1 0 0    (addend)
       ─────────────────────────
                     0 1 1 0 1    (sum = + 13)
```

Note that the sign bits of the AUGEND and ADDEND are both 0 and the sign bit of the sum is 0 indicating that the sum is positive.

Also note that the augend and the addend are made to have the same number of bits. This must always be done in the 2's complement system.

b. Consider the addition of +9 and –4. Remember that the –4 will be in its 2's complement form. Thus, +4 (00100) must be converted to –4 (11100).

```
                          sign bits
        +9           0 1 0 0 1    (augend)
        -4           1 1 1 0 0    (addend)
       ─────────────────────────
                   1 0 0 1 0 1
                   ↑
                   └ this carry is disregarded, so the result is

                     0 0 1 0 1    (sum = + 5)
```

In this case the sign bit of the addend is 1. Note the sign bits also participate in the addition process. In fact, a carry is generated in the last position of addition. THIS CARRY IS ALWAYS DISREGARDED, so the final sum is 00101, which is equivalent to +5.

c. Consider the addition of –9 and +4:

```
        -9 ────→ 1 0 1 1 1
        +4 ────→ 0 0 1 0 0
       ─────────────────────
                 1 1 0 1 1    (sum = -5)
```

The sum in this example has a sign bit of 1, thus indicating a negative number. Since the sum is negative, it is in its 2's – complement form, then the binary sum (1 1 0 1 1) represents the 2's complement of 00101 (decimal 5). Hence 11011 is the equivalent to –5, the correct expected result.

d. Two negative numbers

```
        -9 ────→ 1 0 1 1 1
        -4 ────→ 1 1 1 0 0
       ─────────────────────
               1 1 0 0 1 1
               ↑
               └ this carry is disregarded, so the final result is 10011 (sum = -13)
```

This final result is again negative and in the 2's complement form with a sign bit of 1. ie. 10011 = 01101 (2's comp.) =)13.

e. Equal and opposite numbers

```
 -9 ───→ 1 0 1 1 1
 +9 ───→ 0 1 0 0 1
 ──────────────────
 0     1 0 0 0 0 0
       └ disregarded, so the result is 00000 (sum = +0)
```

The result is obviously +0, as expected.

SUBTRACTION OF TWO'S COMPLEMENT NUMBERS

The subtraction operation using the two's complement system actually involves the operation of addition and is really no different than the various examples considered so far. When subtracting one binary number (the subtrahend) from another binary number (the minuend), the procedure is as follows:

1. Take the 2's complement of the subtrahend, including the sign bit. If the subtrahend is a positive number, this will change it to a negative number in 2's complement form. If the subtrahend is a negative number, this will change it to a positive number in true binary form. In other words, we are changing the sign of the subtrahend.

2. After taking the 2's complement of the subtrahend, it is ADDED to the minuend. The minuend is kept in its original form. The result of this addition represents the required DIFFERENCE. The sign bit of this difference determines whether it is + or – and whether it is in true binary form or 2's complement form.

   Consider the case where +4 is to be subtracted from +9.

   ```
      minuend (9) ── 0 1 0 0 1
   subtrahend (4) ── 0 0 1 0 0
   ```

   Change the subtrahend to its 2's complement form (11100). Now add this to the minuend:

   ```
       0 1 0 0 1   (+9)
   +   1 1 1 0 0   (–4)
     ───────────
     1 0 0 1 0 1
     └ disregard so the result is 00101 = + 5
   ```

   When the subtrahend is changed to its 2's complement form it actually becomes –4, so we are in fact *ADDING* +9 and –4, which is the same as *SUBTRACTING* +4 from +9. (example b.).

CODES AND CODING

GENERAL

Having used decimal numbers for many years, we would like to keep using them. Digital systems, however, force us to use binary numbers. Fortunately, we can compromise by using binary-coded decimals (B.C.D.). These codes combine features of decimal and binary numbers. There are an enormous number of B.C.D. codes. In this section we will discuss only the more common of them.

THE 8421 CODE (B.C.D.)

This code is sometimes referred to as the Natural B.C.D. code since the decimal numbers are represented by the binary code group as follows:

Decimal No.	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

(Binary weightings)

The column returns to 0 after 9. Therefore only 10 of the possible 16 combinations are used.

The decimal number 659 becomes after encoding:

6	5	9
0110	0101	1001

Note that unlike the octal and hexidecimal codes, the binary digits are not pushed together to form a complete binary number, but instead, are retained in their four-bit groups. To put the groups together as for octal or hexidecimal will result in a meaningless group of binary digits. The straight binary representation of 659_{10} is 1010010011_2 (ie. the BCD code group requires 12 bits whereas straight binary only requires 10 bits to represent 659_{10}.

The main advantage of the B.C.D. code is the relative ease of converting to and from decimal.

Many other four-bit codes exist and the following tables show some of the more common B.C.D. codes.

Decimal	7 4 2 1	5 4 2 1	5 3 1 1	4 2 2 1	$8\,4\,\overline{2}\,\overline{1}$
0	0000	0000	0000	0000	0000
1	0001	0001	0001	0001	0111
2	0010	0010	0011	0010	0110
3	0011	0011	0100	0011	0101
4	0100	0100	0101	1000	0100
5	0101	1000	1000	0111	1011
6	0110	1001	1001	1100	1010
7	1000	1010	1011	1101	1001
8	1001	1011	1100	1110	1000
9	1010	1100	1101	1111	1111

The $8\,4\,\overline{2}\,\overline{1}$ code has negative weightings in some of the columns and therefore must be subtracted from the total for each decimal digit. The $8\,4\,\overline{2}\,\overline{1}$ and 4 2 2 1 codes are self complementing codes (ie. the numbers 1_{10} and 8_{10} are the complement of each other. The choice of code depends on the purpose of the circuit, the application of the device and the adjoining circuits. The 8 4 2 1 code however, is the most commonly used code. It is for this reason that when the B.C.D. code is discussed, the 8 4 2 1 code is assumed.

Decade counters were discussed in an earlier section so will not be discussed here.

B.C.D. ADDITION

A disadvantage of the 8 4 2 1 code is that the rules for binary addition do not apply to the entire 8 4 2 1 number, but only to the individual 4-bit groups, eg. adding 12 and 9 in straight binary is easy:

$$\begin{array}{r} 12 \\ +\ 9 \\ \hline 21 \end{array} \qquad \begin{array}{r} 1100 \\ +1001 \\ \hline 10101 \end{array}$$

If we try this in the 8 4 2 1 code, we get an unacceptable answer.

$$\begin{array}{rrrr} 8\,4\,2\,1 = & 12 & 0001 & 0010 \\ & +\ 9\ + & & 1001 \\ \hline & 21 & 0001 & 1011 \end{array}$$

We are unable to decode 0001, 1011 because 1011 does not exist in the 8 4 2 1 code. Remember the largest 8 4 2 1 code group is 1001 (9). Therefore, the addition of 8 4 2 1 numbers is not so simple as for binary numbers. This means that some method for carrying out B.C.D. addition must be found.

One method of adding B.C.D. numbers is given in the following examples.

a. Sum Equals Nine or Less

$$\begin{array}{r} 5 \\ +\ 4 \\ \hline 9 \end{array} \qquad \begin{array}{r} 0101 \\ +0100 \\ \hline 1001 \end{array}$$

0101 —— B.C.D. for 5
+0100 —— " " 4
1001 —— " " 9

The addition is carried out as in normal binary addition and the sum is 1001, which is the B.C.D. code for 9. As another example 45 + 33.

$$\begin{array}{rrl} 45 & 0100\ \ 0101 & \text{— B.C.D. for 45} \\ +\ 33 & +\ 0011\ \ 0011 & \text{— " " 33} \\ \hline 78 & 0111\ \ 1000 & \text{— " " 78} \end{array}$$

Here, none of the sums of the pairs of decimal digits exceeds nine, therefore, NO DECIMAL CARRIES WERE PRODUCED. For these cases the B.C.D. addition process is straight forward and is actually the same as binary addition.

b. Sum Greater Than Nine

$$\begin{array}{rrl} 6 & 0110 & \text{— B.C.D. for 6} \\ +\ 7 & +\ 0111 & \text{— " " 7} \\ \hline 13 & 1101 & \text{— Invalid code group for B.C.D.} \end{array}$$

The sum 1101 does not exist in the B.C.D. code; it is one of the six forbidden or invalid 4-bit code groups. This has occurred because the sum of the two digits exceeds 9. Whenever this occurs the sum has to be corrected by the addition of six (0110) to take into account the skipping of the six invalid code groups:

$$\begin{array}{rrrl} 6 & & 0110 & \text{— B.C.D. for 6} \\ +\ 7 & & +\ 0111 & \text{— " " 7} \\ \hline 13 & & 1101 & \text{— Invalid sum} \\ & & +\ 0110 & \text{— Add 6 for correction} \\ \hline & \underbrace{0001}_{1} & \underbrace{0011}_{3} & \text{— B.C.D. for 13} \end{array}$$

As shown above, 0110 is added to the invalid sum and produces the correct B.C.D. result. Note that a carry is produced into the second decimal position. This addition of 0110 has to be performed whenever the sum of the two decimal digits is greater than 9.

As another example, 47 + 35 in B.C.D.

$$\begin{array}{rrrl} 47 & 0100 & 0111 & \text{— B.C.D. for 47} \\ +\ 35 & +0011 & 0101 & \text{— " " 35} \\ \hline 82 & 0111 & 1100 & \text{— Invalid sum in first digit} \\ & & 0110 & \text{— Add 6} \\ \hline & \underbrace{1000}_{8} & \underbrace{0010}_{2} & \text{— Correct B.C.D. sum} \end{array}$$

The addition of the 4-bit codes for the 7 and 5 digits results in an invalid sum and is corrected by adding 0110. Note that this generates a carry of 1, which is carried over to be added to the B.C.D. sum of the second-position digits.

To summarize the B.C.D. addition procedure:

1. Add, using ordinary binary addition, the B.C.D. code groups for each digit position.
2. For those positions where the sum is 9 or less, no correction is needed. The sum is in proper B.C.D. form.
3. When the sum of two digits is greater than 9, a correction of 0110 should be added to that sum to get the proper B.C.D. result. This will always produce a carry into the next decimal position.

   The procedure for B.C.D. addition is clearly more complicated than straight binary addition. This is true for other B.C.D. arithmetic operations.

THE EXCESS – 3 CODE

The EXCESS–3 code is related to the B.C.D. code and is sometimes used instead of B.C.D. because it possesses advantages in certain arithmetic operations. The XS–3 code for a decimal number is performed in the same manner as B.C.D. except that 3 is added to EACH decimal digit before encoding it in binary. For example, to encode the decimal number 4 into XS–3 code, we must first add 3 to obtain 7. Then the 7 is encoded in its equivalent 4-bit binary code 0111.

As another example, let us convert 57_{10} into its XS–3 code

$$\begin{array}{cccl} 5 & & 7 & \\ \underline{+\ 3} & & \underline{+\ 3} & \text{add 3 to each digit} \\ 8 & & 10 & \\ \downarrow & & \downarrow & \\ 1000 & & 1010 & \text{convert to 4-bit binary code} \end{array}$$

The following table lists the B.C.D. and XS–3 code representations for the decimal digits. Note that both codes use only 10 of the 16 possible 4-bit code groups. The XS–3 code, however, does not use the same code groups. For XS–3, the invalid code groups are: 0000, 0001, 0010, 1101, 1110 and 1111.

Decimal	B.C.D.	XS–3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

10.5 THE GREY CODE

The GREY CODE belongs to a class of codes called MINIMUM CHANGE CODES, in which only ONE bit in the code group changes when going from one step to the next. The Grey-

Code is an unweighted code, meaning that the bit positions in the code groups do not have any specific weight assigned to them. Because of this, the Grey-Code is not suited for arithmetic operations but finds applications in input/output devices and some types of analogue to digital converters.

The following table lists the Grey-Code representations for the decimal numbers 0 to 15, together with the straight binary code.

Decimal	Binary Code	Grey Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

The Grey-Code is often used in situations where other codes, such as binary, might produce erroneous or ambiguous results during those transitions in which more than one bit of the code is changing, eg. using the binary code and going from 0111 to 1000 requires that all four bits change simultaneously. Depending on the device or circuit that is generating the bits, there may be a significant difference in the transition times of the different bits. If so, the transition from 0111 to 1000 could produce one or more intermediate states. For example, if the MSB changes faster than the rest, the following transitions could occur:

0 1 1 1 ——— decimal 7

1 1 1 1 ——— erroneous code

1 0 0 0 ——— decimal 8

The occurrence of 1111 is only momentary but it could conceivably produce erroneous operation of the elements that are being controlled by the bits. Obviously, using the Grey-Code would eliminate this problem, since only one bit change occurs per transition and no "race" between bits can occur.

CONVERTING FROM BINARY TO GREY-CODE

Any binary number can be converted to its Grey-Code equivalent as follows:

1. The MSB of the binary number is the same for the Grey-Coded number.
2. Exclusive OR each pair of adjacent bits to obtain the next grey bit, eg.

```
1 0 1 1 0 0 1 1 —— binary number
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
1 1 1 0 1 0 1 0 —— grey number
```

This is illustrated in Fig. 10.1.

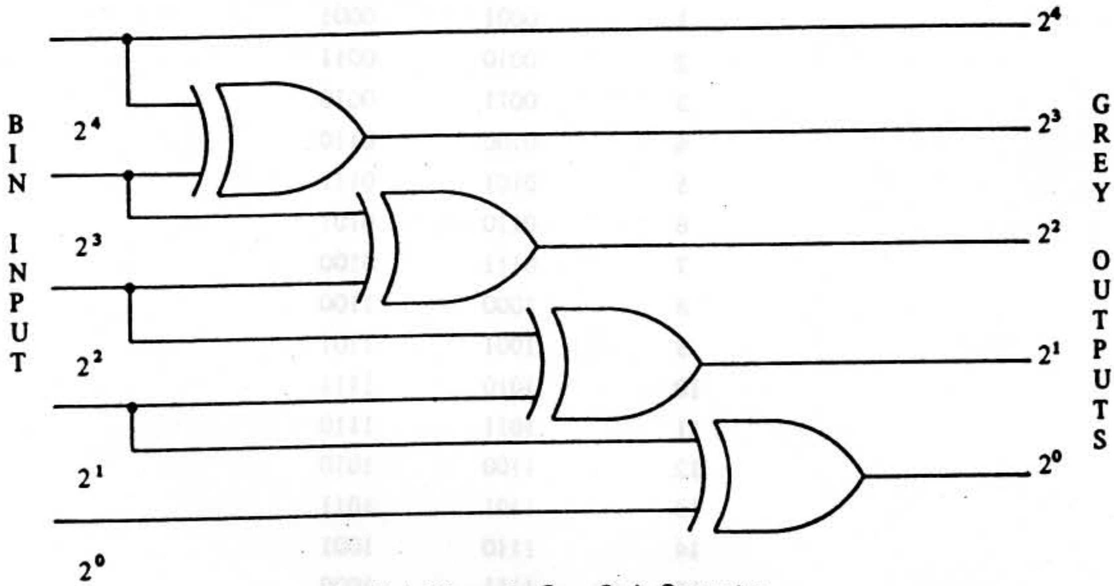


Fig. 10.1 Binary to Grey-Code Converter

CONVERTING FROM GREY-CODE TO BINARY

Any Grey-Coded number can be converted to its binary equivalent as follows:

1. The MSB of the Grey number is the same for the binary number.
2. Exclusive OR diagonally from bottom to top as shown to obtain the next binary bit, eg.

```
1 1 0 0 1 0 1 1    Grey-Coded number

1 0 0 0 1 1 0 1    Binary number
```

This can be illustrated in Fig. 10.2.

ALPHANUMERIC CODES

We have studied several codes that are used to represent numerical data, that is, numbers. Many digital systems, such as the computer, also use alphabetic data (letters) and special characters (punctuation and mathematics symbols) in addition to numbers. Many codes have been devised for representing letters, characters and numbers. Such codes are called ALPHANUMERIC CODES.

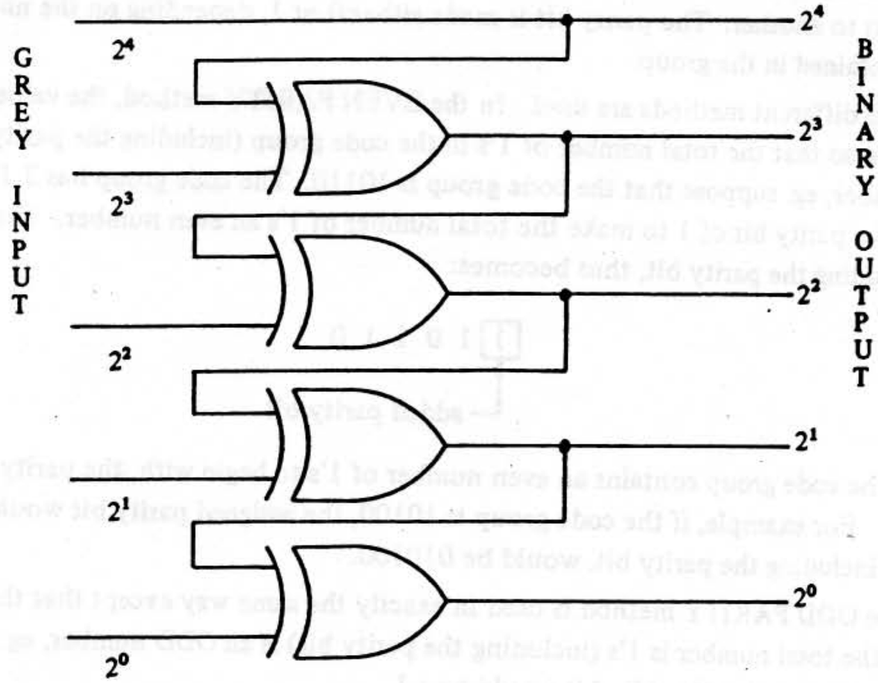


Fig. 10.2 Grey to Binary Code Converter

One such code that has been devised is the ASCII (*A*merican *S*tandard *C*ode for *I*nformation *I*nterchange. Pronounced Ask−−ee.), which is used in the transmission of digital information. The ASCII shown in Appendix B table has 7 bits, which indicates that it can represent $2^7 = 128$ different characters. Only some of these are shown in Appendix B. For example, using this code, the statement "PAY = $5.00" would be stored as:

1010000	1000001	1011001	0111101	0100100	0110101	Binary
50	41	59	3D	24	35	Hexidecimal
P	A	Y	=	$	5	Character

PARITY METHOD FOR ERROR DETECTION

The transmission of binary data from one location to another is common-place in all digital systems. Listed below are just some examples of this:

1. **Binary data output from a computer being recorded on magnetic tape.**
2. **Transmission of binary data over telephone lines, such as between a computer and a remote console.**
3. **A number is taken from the computer memory and placed in the arithmetic unit, where it is to be operated on and the sum placed back into memory.**

The process of transferring data is subject to error, although modern equipment has been designed to reduce the probability of error. However, even relatively infrequent errors can cause useless results, so it is desirable to detect them whenever possible. One of the most widely used schemes for error detection is the PARITY method.

THE PARITY BIT

A parity bit is an extra bit that is attached to a code group which is being transferred from one location to another. The parity bit is made either 0 or 1, depending on the number of 1's that are contained in the group.

Two different methods are used. In the EVEN PARITY method, the value of the parity bit is chosen so that the total number of 1's in the code group (including the parity bit) is an EVEN number, eg. suppose that the code group is 10110. The code group has 3 1's, therefore, we will add a parity bit of 1 to make the total number of 1's an even number. The NEW code group, including the parity bit, thus becomes:

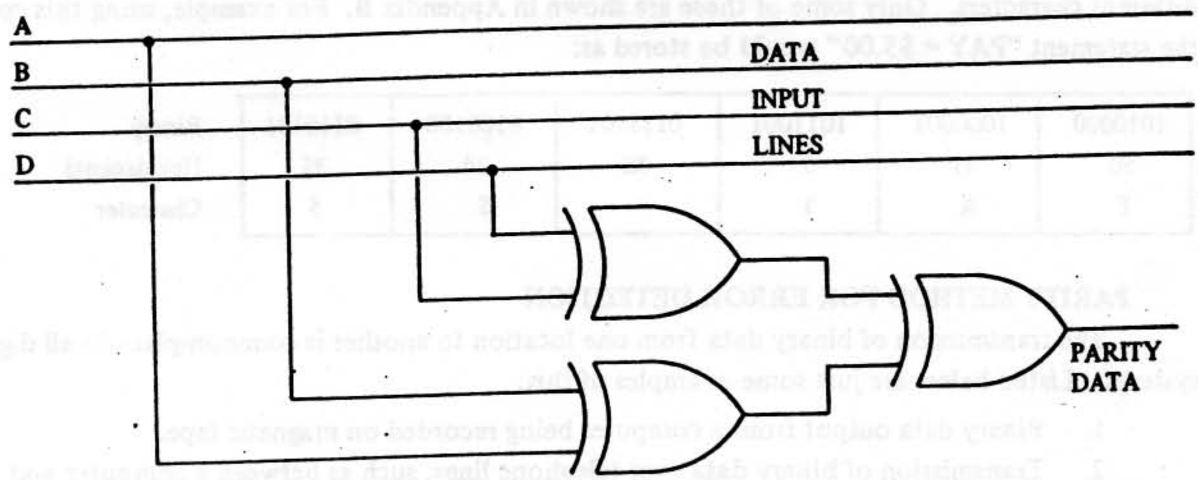
$\boxed{1}$ 1 0 1 1 0

added parity bit

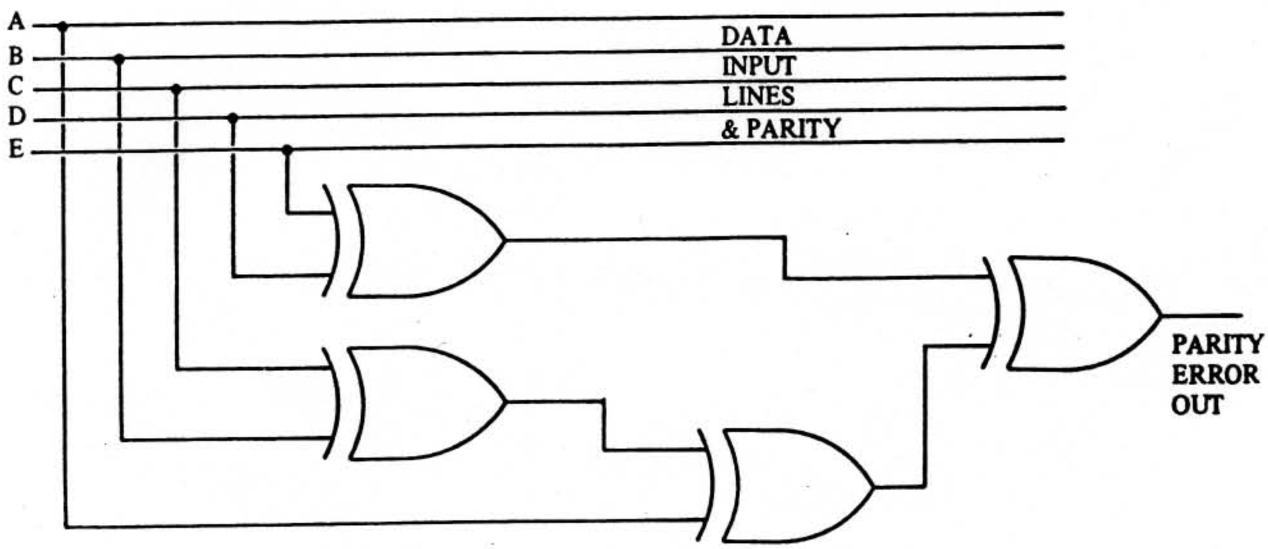
If the code group contains an even number of 1's to begin with, the parity bit is given a value of 0. For example, if the code group is 10100, the assigned parity bit would be 0, so the new code, including the parity bit, would be 010100.

The ODD PARITY method is used in exactly the same way except that the parity bit is chosen so the total number is 1's (including the parity bit) is an ODD number, eg. for the code group 01100, the assigned parity bit would be a 1.

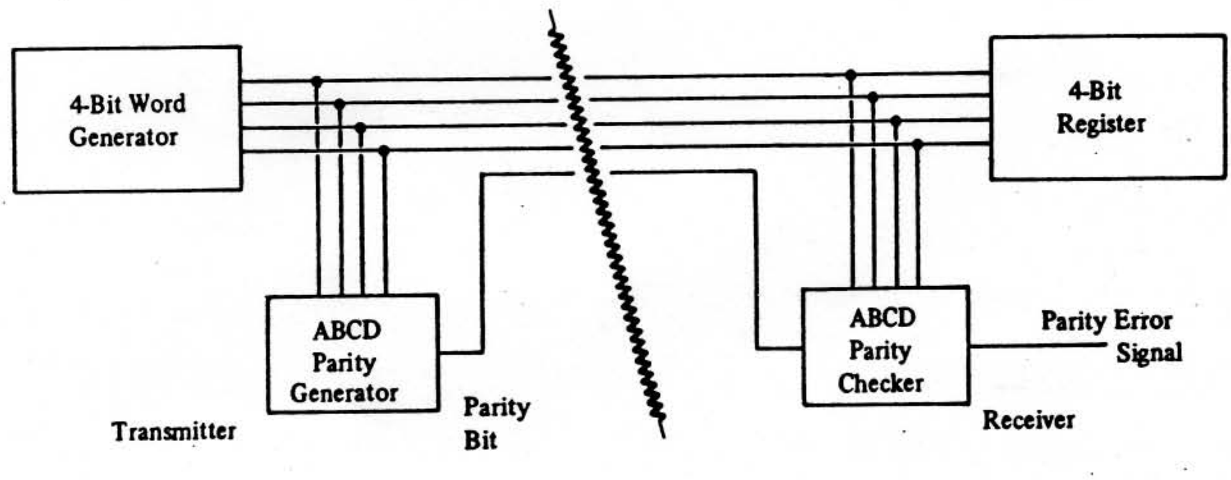
Regardless of whether even parity or odd parity is used, the parity bit is added to the code word and is transmitted as part of the code word. Fig. 10.3 shows how the parity bit is generated and then subsequently checked for an even parity system.



(a)



(b)

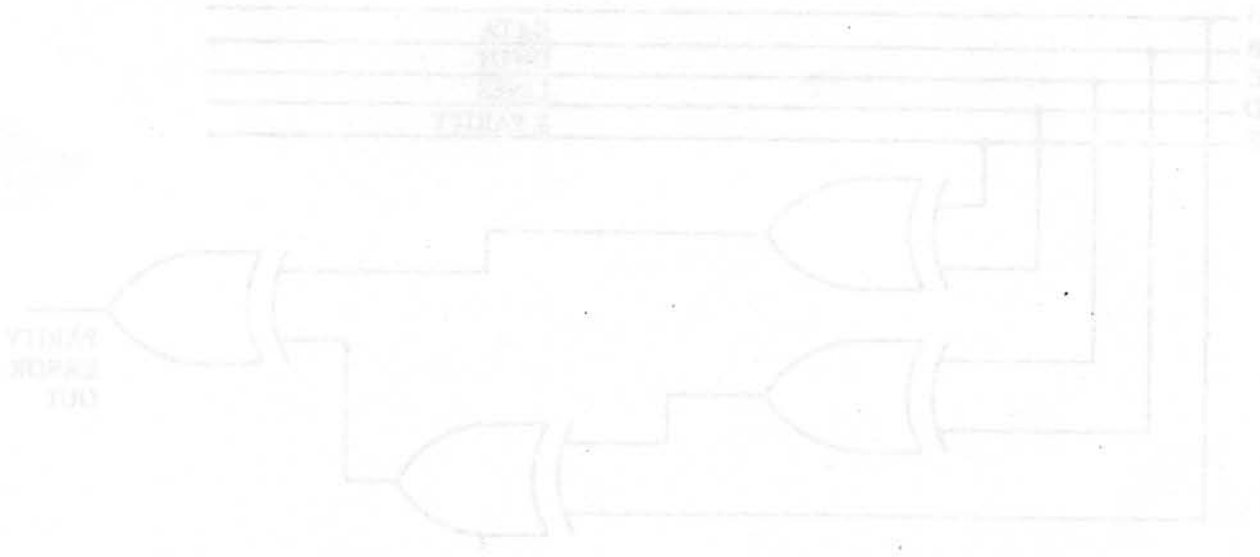


(c)

Fig. 10.3 (a) An EVEN Parity Generator

(b) An EVEN Parity Checker

(c) General Block Diagram of the Parity Met. od



(b)

(c)

Fig. 10.9 (a) An EVEN Parity Generator
(b) An EVEN Parity Checker
(c) General Block Diagram of the Parity Method

Base Conversions

The following table lists base conversions for all one-byte values.

DEC.	BINARY	HEX.	OCT.
0	00000000	00	000
1	00000001	01	001
2	00000010	02	002
3	00000011	03	003
4	00000100	04	004
5	00000101	05	005
6	00000110	06	006
7	00000111	07	007
8	00001000	08	010
9	00001001	09	011
10	00001010	0A	012
11	00001011	0B	013
12	00001100	0C	014
13	00001101	0D	015
14	00001110	0E	016
15	00001111	0F	017
16	00010000	10	020
17	00010001	11	021
18	00010010	12	022
19	00010011	13	023
20	00010100	14	024
21	00010101	15	025
22	00010110	16	026
23	00010111	17	027
24	00011000	18	030
25	00011001	19	031
26	00011010	1A	032
27	00011011	1B	033
28	00011100	1C	034
29	00011101	1D	035
30	00011110	1E	036
31	00011111	1F	037
32	00100000	20	040
33	00100001	21	041
34	00100010	22	042
35	00100011	23	043
36	00100100	24	044
37	00100101	25	045
38	00100110	26	046
39	00100111	27	047
40	00101000	28	050
41	00101001	29	051
42	00101010	2A	052
43	00101011	2B	053
44	00101100	2C	054
45	00101101	2D	055
46	00101110	2E	056
47	00101111	2F	057
48	00110000	30	060
49	00110001	31	061
50	00110010	32	062
51	00110011	33	063
52	00110100	34	064
53	00110101	35	065
54	00110110	36	066
55	00110111	37	067
56	00111000	38	070
57	00111001	39	071
58	00111010	3A	072
59	00111011	3B	073
60	00111100	3C	074
61	00111101	3D	075
62	00111110	3E	076
63	00111111	3F	077
64	01000000	40	100
65	01000001	41	101
66	01000010	42	102
67	01000011	43	103
68	01000100	44	104
69	01000101	45	105
70	01000110	46	106
71	01000111	47	107
72	01001000	48	110
73	01001001	49	111
74	01001010	4A	112
75	01001011	4B	113
76	01001100	4C	114
77	01001101	4D	115
78	01001110	4E	116
79	01001111	4F	117
80	01010000	50	120
81	01010001	51	121
82	01010010	52	122
83	01010011	53	123
84	01010100	54	124
85	01010101	55	125

DEC.	BINARY	HEX.	OCT.
86	01010110	56	126
87	01010111	57	127
88	01011000	58	130
89	01011001	59	131
90	01011010	5A	132
91	01011011	5B	133
92	01011100	5C	134
93	01011101	5D	135
94	01011110	5E	136
95	01011111	5F	137
96	01100000	60	140
97	01100001	61	141
98	01100010	62	142
99	01100011	63	143
100	01100100	64	144
101	01100101	65	145
102	01100110	66	146
103	01100111	67	147
104	01101000	68	150
105	01101001	69	151
106	01101010	6A	152
107	01101011	6B	153
108	01101100	6C	154
109	01101101	6D	155
110	01101110	6E	156
111	01101111	6F	157
112	01110000	70	160
113	01110001	71	161
114	01110010	72	162
115	01110011	73	163
116	01110100	74	164
117	01110101	75	165
118	01110110	76	166
119	01110111	77	167
120	01111000	78	170
121	01111001	79	171
122	01111010	7A	172
123	01111011	7B	173
124	01111100	7C	174
125	01111101	7D	175
126	01111110	7E	176
127	01111111	7F	177
128	10000000	80	200
129	10000001	81	201
130	10000010	82	202
131	10000011	83	203
132	10000100	84	204
133	10000101	85	205

DEC.	BINARY	HEX.	OCT.
134	10000110	86	206
135	10000111	87	207
136	10001000	88	210
137	10001001	89	211
138	10001010	8A	212
139	10001011	8B	213
140	10001100	8C	214
141	10001101	8D	215
142	10001110	8E	216
143	10001111	8F	217
144	10010000	90	220
145	10010001	91	221
146	10010010	92	222
147	10010011	93	223
148	10010100	94	224
149	10010101	95	225
150	10010110	96	226
151	10010111	97	227
152	10011000	98	230
153	10011001	99	231
154	10011010	9A	232
155	10011011	9B	233
156	10011100	9C	234
157	10011101	9D	235
158	10011110	9E	236
159	10011111	9F	237
160	10100000	A0	240
161	10100001	A1	241
162	10100010	A2	242
163	10100011	A3	243
164	10100100	A4	244
165	10100101	A5	245
166	10100110	A6	246
167	10100111	A7	247
168	10101000	A8	250
169	10101001	A9	251
170	10101010	AA	252
171	10101011	AB	253
172	10101100	AC	254
173	10101101	AD	255
174	10101110	AE	256
175	10101111	AF	257
176	10110000	B0	260
177	10110001	B1	261
178	10110010	B2	262
179	10110011	B3	263
180	10110100	B4	264
181	10110101	B5	265
182	10110110	B6	266

DEC.	BINARY	HEX.	OCT.
183	10110111	B7	267
184	10111000	B8	270
185	10111001	B9	271
186	10111010	BA	272
187	10111011	BB	273
188	10111100	BC	274
189	10111101	BD	275
190	10111110	BE	276
191	10111111	BF	277
192	11000000	C0	300
193	11000001	C1	301
194	11000010	C2	302
195	11000011	C3	303
196	11000100	C4	304
197	11000101	C5	305
198	11000110	C6	306
199	11000111	C7	307
200	11001000	C8	310
201	11001001	C9	311
202	11001010	CA	312
203	11001011	CB	313
204	11001100	CC	314
205	11001101	CD	315
206	11001110	CE	316
207	11001111	CF	317
208	11010000	D0	320
209	11010001	D1	321
210	11010010	D2	322
211	11010011	D3	323
212	11010100	D4	324
213	11010101	D5	325
214	11010110	D6	326
215	11010111	D7	327
216	11011000	D8	330
217	11011001	D9	331
218	11011010	DA	332

DEC.	BINARY	HEX.	OCT.
219	11011011	DB	333
220	11011100	DC	334
221	11011101	DD	335
222	11011110	DE	336
223	11011111	DF	337
224	11100000	E0	340
225	11100001	E1	341
226	11100010	E2	342
227	11100011	E3	343
228	11103100	E4	344
229	11100101	E5	345
230	11100110	E6	346
231	11100111	E7	347
232	11101000	E8	350
233	11101001	E9	351
234	11101010	EA	352
235	11101011	EB	353
236	11101100	EC	354
237	11101101	ED	355
238	11101110	EE	356
239	11101111	EF	357
240	11110000	F0	360
241	11110001	F1	361
242	11110010	F2	362
243	11110011	F3	363
244	11110100	F4	364
245	11110101	F5	365
246	11110110	F6	366
247	11110111	F7	367
248	11111000	F8	370
249	11111001	F9	371
250	11111010	FA	372
251	11111011	FB	373
252	11111100	FC	374
253	11111101	FD	375
254	11111110	FE	376
255	11111111	FF	377

From: Holbrook Submarine Museum
ID: PB 1377
Location: Holbrook Submarine Museum mezzanine floor in box 35
Scanned: 20221228
Scanned By: Dirk Stoffels
Doc Source: Royal Australian Navy Submarine School
Doc Marking: POETS Davis
Scanning Notes: Pages edges were placed square to scanner guides. Any image skew comes from original page skew.